Integrating Custom Formulas into Formula Builder in SAP BI 7.0

Applies to:

SAP Net Weaver BW 7.0. For more information, visit the EDW homepage

Summary

Detailed description on how to integrate custom formulas into BW's Formula Builder

Author: Adlin Sundararaj

Company: Accenture Services Pvt. Ltd.

Created on: 19 July, 2011

Author Bio

Adlin Sundararaj is working as SAP-BI Consultant in Accenture Services Private Ltd and has got rich experience on end to end implementation as well as maintenance projects in BW 3.5 & BW 7.0.

© 2011 SAP AG

Table of Contents

Introduction 3
Formula Builders in SAP BW: 3
Business Scenario 3
Step –by –Step Solution 3
Step 1 - Creating Implementation for BADI RSAR_CONNECTOR 4
Step 2 – Creating Custom Formula 6
Step 3 – Integrating Custom Methods into BADI implementation 9
Step 4 – Checking in Formula Builder 11
Conclusion 12
Related Content 13
Disclaimer and Liability Notice 14

© 2011 SAP AG

Introduction

In SAP BW, we have an option to choose formula during transformation. This document shows how we add custom formulas into the formula builder so that custom formulas can be assigned to the fields while mapping.

This helps us to avoid ABAP code often in the transformation and provides reusability.

Formula Builders in SAP BW:

The transformation library, in collaboration with the formula builder, enables to easily create formulas, without using ABAP coding.

The transformation library has over 70 pre-defined functions, in the following categories:

- Functions for character strings
- Date functions
- Basic functions
- Mathematical functions
- Suitable functions
- Miscellaneous functions

We have the option to implement self-defined functions in the transformation library of the formula builder. We can integrate existing function modules in these self-defined functions. In doing so, we can also make available special functions to be used frequently that are not contained in the transformation library.

The formula builder has two modes: Standard and expert mode. In the standard mode, we can only enter the formulas using the pushbuttons and by double clicking on functions and fields. In the expert mode, however, we can enter formulas directly. we can also toggle between the tow modes when entering a formula.

Business Scenario

A customer has a requirement to transform data coming from Source system into the data warehouse, and consistently uses the same data transformation.

In this example, we are going to transform a timestamp into a date field.

Instead of continuously writing the same coding in ABAP routines, the customer can create a reusable formula.

Step –by –Step Solution

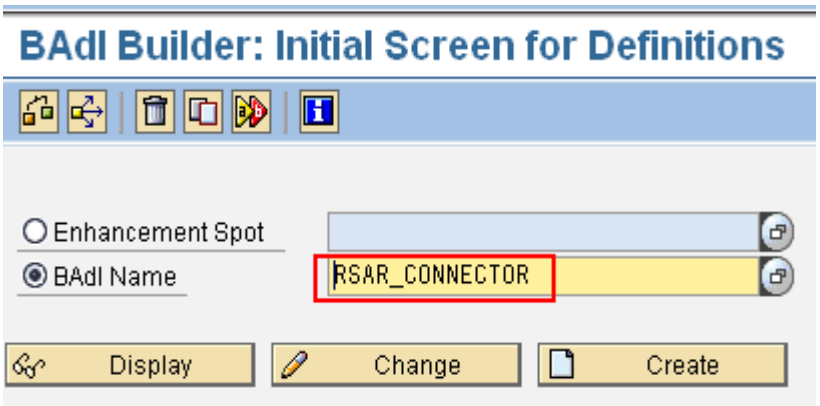
Custom formulas are introduced into the transformation library of the Formula Builder by means of BAdi (Business Add-In) RSAR_CONNECTOR.

The step by step solution that follows will demonstrate how to achieve this.

© 2011 SAP AG

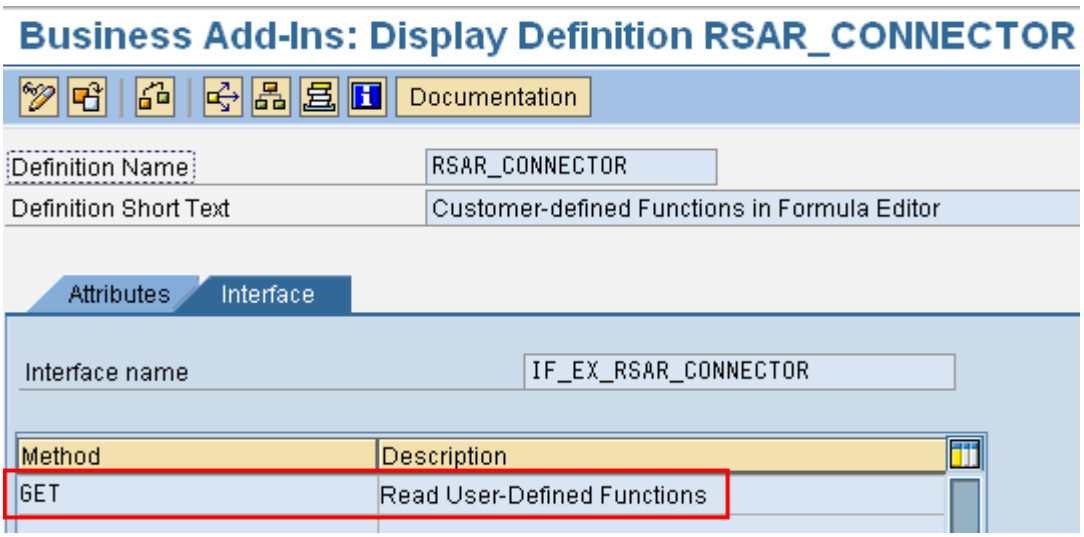
Step 1 - Creating Implementation for BADI RSAR_CONNECTOR

1. Go to T code se18; give the BADI name as RSAR_CONNECTOR → display



You will see a method 'GET' in the interface tab of the BADI.

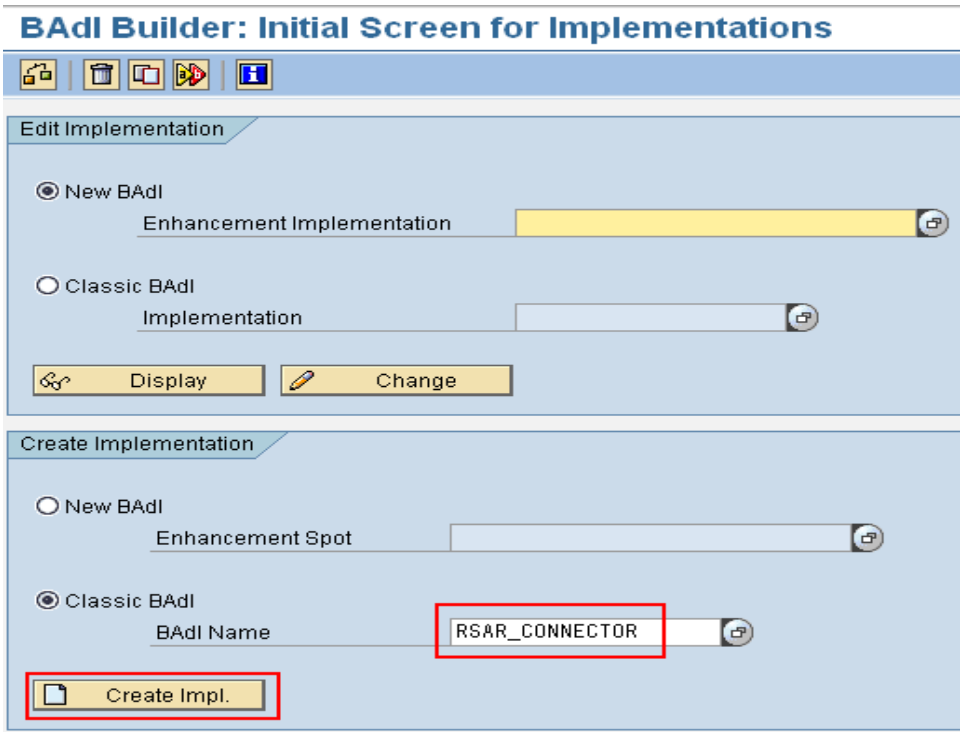
The 'GET' method is the key to the solution. It is called when you list all the formulas that are available. We will enter all our custom methods within the 'GET' method making them available at runtime to the BI developer.



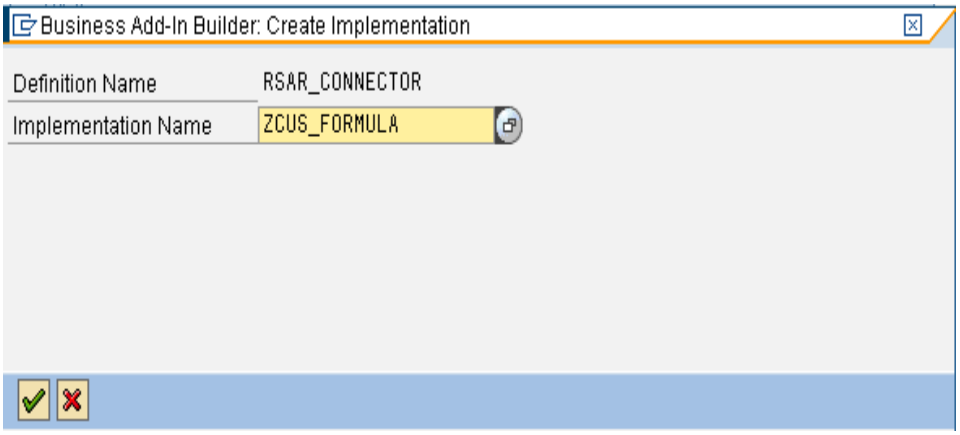
© 2011 SAP AG

2. To create an implementation for this BADI, go to T code se19;

   In the 'Create Implementation' section, change the radio button to 'Classic BAdi' and enter 'RSAR_CONNECTOR' as the BAdi name and click on Create Impl. button



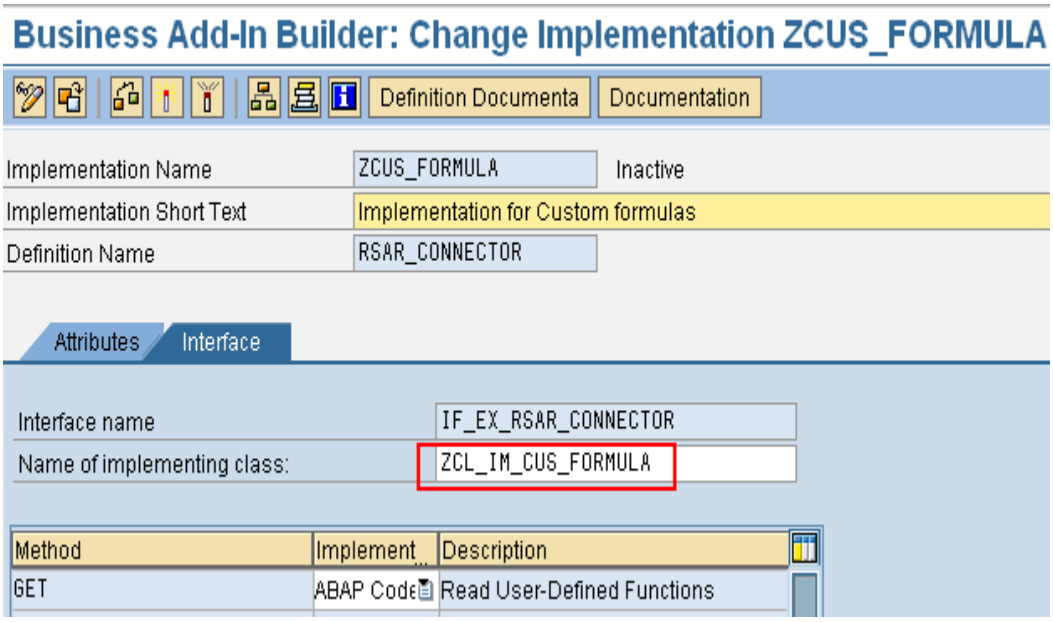
3. Give the implementation a suitable name for e.g. ZCUS_FORMULA, and then press the continue button .



© 2011 SAP AG

4. This will take you to the definition of the implementation.

   Make note of the name of the implementing class you can see in the screenshot (ZCL_IM_CUS_FORMULA), we will use this later to store our methods.

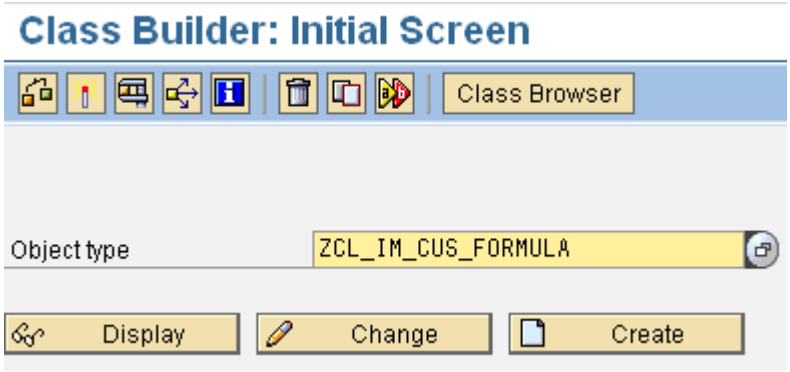


5. Press the activate button on the toolbar in order to activate your implementation and its underlying objects.

6. Assign your implementation to a transportable package and include it in a transport request.

Step 2 – Creating Custom Formula

1. We need code our logic inside class ZCL_IM_CUS_FORMULA; to do this, Go to transaction SE24 in order to edit Class ZCL_IM_CUS_FORMULA.



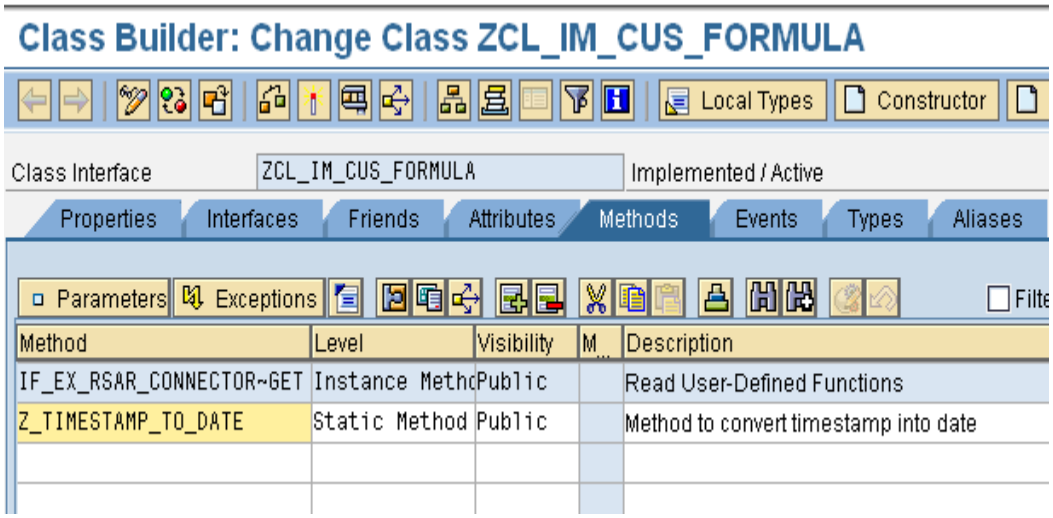
Alternatively, instead of using the class generated from creating the implementation, you could implement your own class with methods.

© 2011 SAP AG

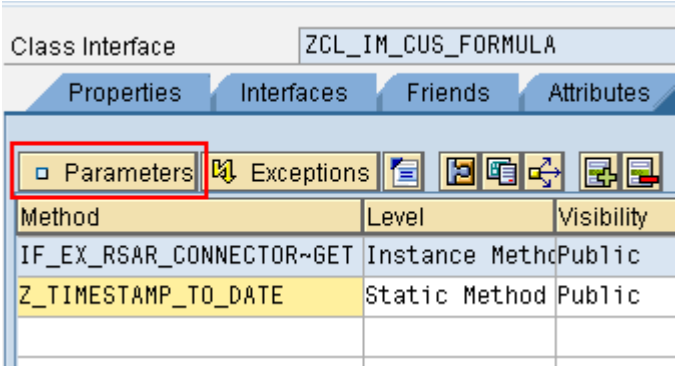
2. Click on change button

3. On the 'Methods' tab we can implement as many custom methods as you wish. In our example, we are going to implement a method to convert the time stamp into date

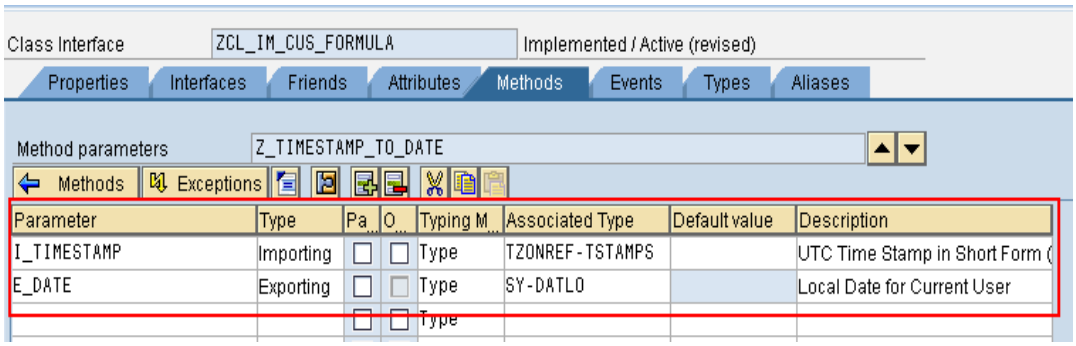
4. Give a meaningful technical name and description to our method, and declare it as Static and Public.



5. Once this is done, proceed to defining its parameters by pressing the 'Parameters' button.



6. Importing refers to the parameter that you pass to the method for execution and exporting refers to the parameter that is passed back once the method has executed.
In our example, we pass an 'Importing' parameter I_TIMESTAMP (timestamp) which is then converted to a calendar day and passed back via exporting parameter E_DATE.



© 2011 SAP AG

Here we have declared 'I_TIMESTAMP' as a type of 'TZONREF-TSTAMPS' which refers to 'UTC Time Stamp in Short Form (YYYYMMDDhhmmss)' and 'E_DATE' as a type of 'SY-DATLO' which refers to 'Local Date for Current User'.

7. Press the 'Method' button

to return back to the main method definition.

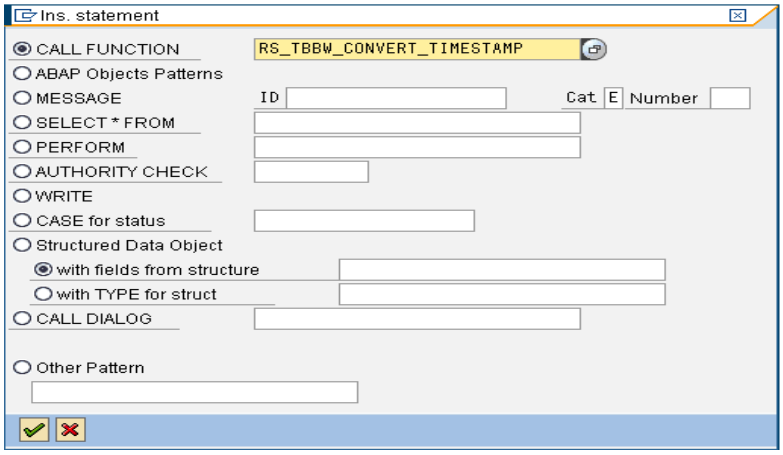
8. Double-click on the method name or press the button to view the method code.

9. Enter code between the sections "METHOD <method name>. ENDMETHOD."

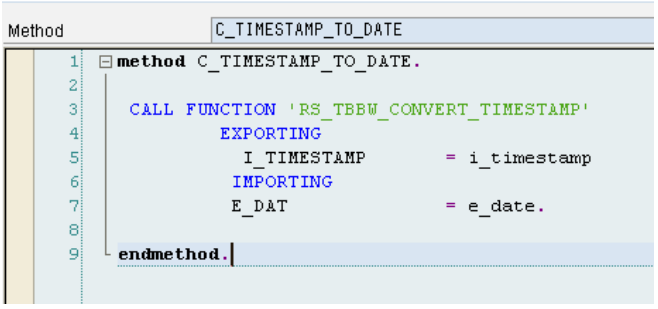
   Remember to use your parameters exactly as you have named them on the parameter definition screen for your method.
   To convert the time stamp into ate we have standard FM defined by SAP.

   Call the FM using Pattern Button

   Give the Function Module name in CALL FUNCTION and press



   Now specify the importing and exporting parameters



10. Activate your method.

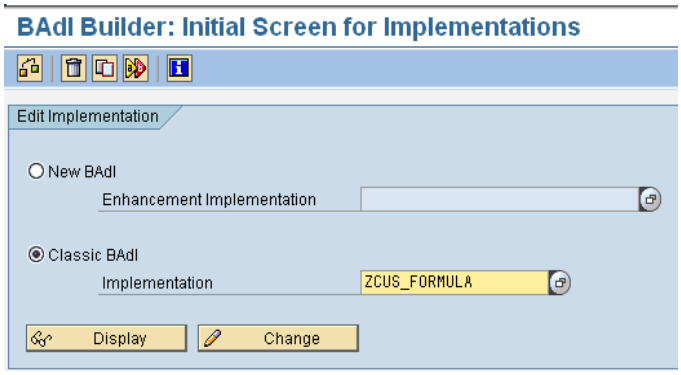
Note: Methods must have the following attributes when you create them:

a) They are declared static and public.
b) They can only have importing, exporting, and returning parameters. Changing parameters are not permitted.
c) They can only have one exporting or returning parameter.
d) Exporting parameters cannot have a generic type.

© 2011 SAP AG

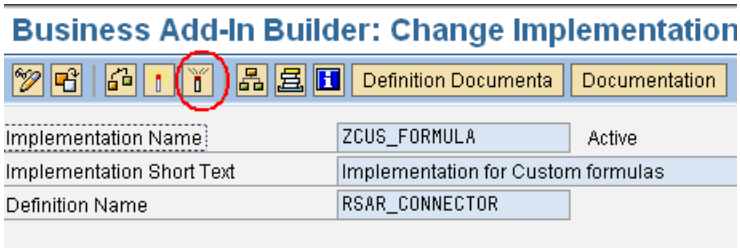
Step 3 – Integrating Custom Methods into BADI implementation

1. Go back to transaction SE19; In the 'Edit Implementation' section, change the radio button to 'Classic BAdi' and enter Implementation name 'ZCUS_FORMULA'.



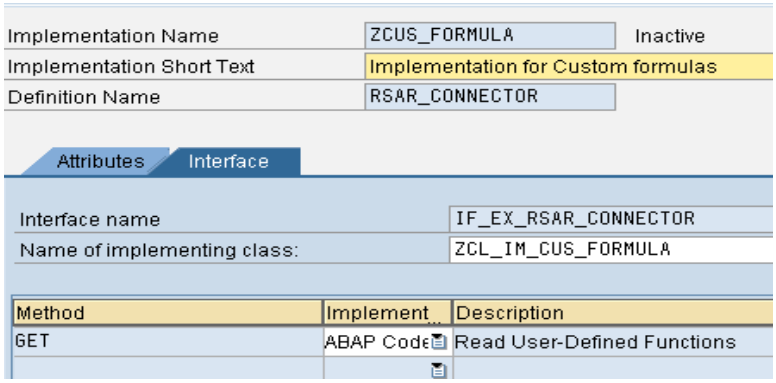
2. Click on change button

3. Make sure to inactivate your BAdi implementation as you cannot edit active implementations



4. Double-click on the 'GET' method on the 'Interface' tab in order to incorporate your custom methods as formulas.

   The 'GET' method allows us to do two things:
   a. Firstly we get to create new categories for custom formulas (optional)
   b. Secondly we get to assign custom formulas to those new groupings or alternatively to the default grouping.



© 2011 SAP AG

5. Initially the method will be empty.

```
Method                    IF_EX_RSAR_CONNECTOR~GET
1  method IF_EX_RSAR_CONNECTOR~GET.
2
3  endmethod.
```

6. The base format of the code should look like the code that follows. The logic is as follows:

   - When i_key is blank, you can use it to define new categories for formulas. In this section appending to c_operands creates a new category.
   - When i_key is populated, it is referring to a category name in which you want to place the custom formulas. In this section appending to c_operands creates a new function within that category.

   The default category is CUSTOM. If you choose not to create your own categories then all custom formulas should be placed in the section when i_key is 'CUSTOM'.

   Code the below in the GET method

```
IF_EX_RSAR_CONNECTOR~GET                                   Active
method IF_EX_RSAR_CONNECTOR~GET.

  Data: l_function type SFBEOPRND.

  Case i_key.

  WHEN 'CUSTOM'.

    CLEAR l_function.
      l_function-tech_name = 'C_TIMESTAMP_TO_DATE'.
      l_function-descriptn = 'Method to convert timestamp into date'.
      l_function-class = 'ZCL_IM_CUS_FORMULA'.
      l_function-method = 'C_TIMESTAMP_TO_DATE'.
      APPEND l_function TO c_operands.

  ENDCASE.
  endmethod.
```

   Each new formula needs the following four fields populated:

   1. descriptn → This is the description that will be displayed within the Formula Builder

   2. tech_name → This is technical name that will uniquely identify the formula in the list of formulas

   3. class → This is the ABAP class in which the formula method is implemented

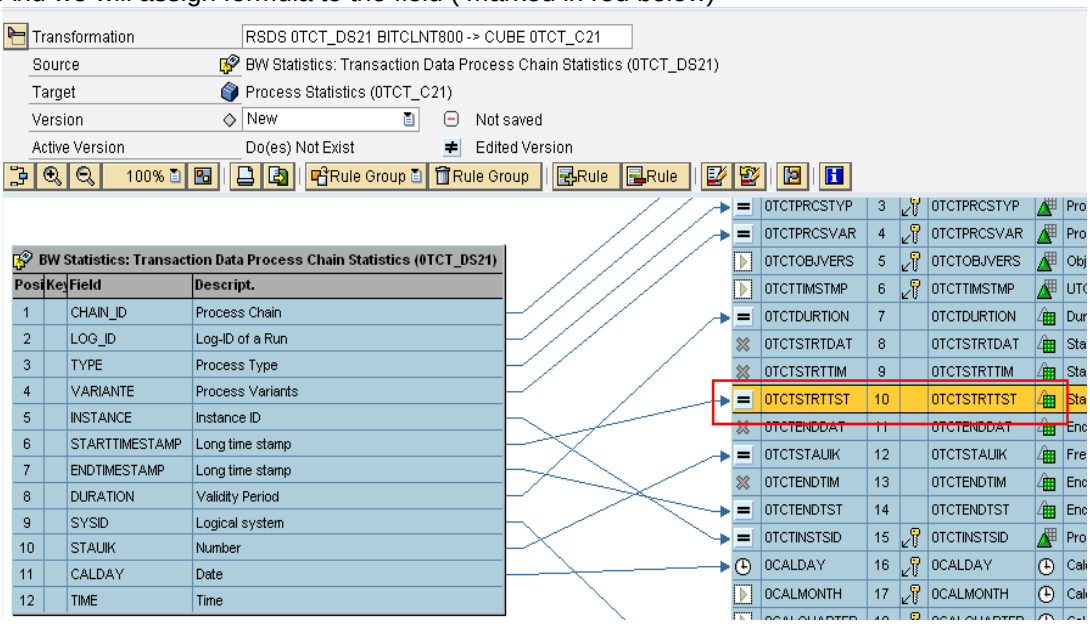
   4. method → This is the method within the ABAP class mentioned above in which the formula is implemented

7. Activate the GET method using the activate button on the toolbar.
8. Go back one screen and activate your implementation.

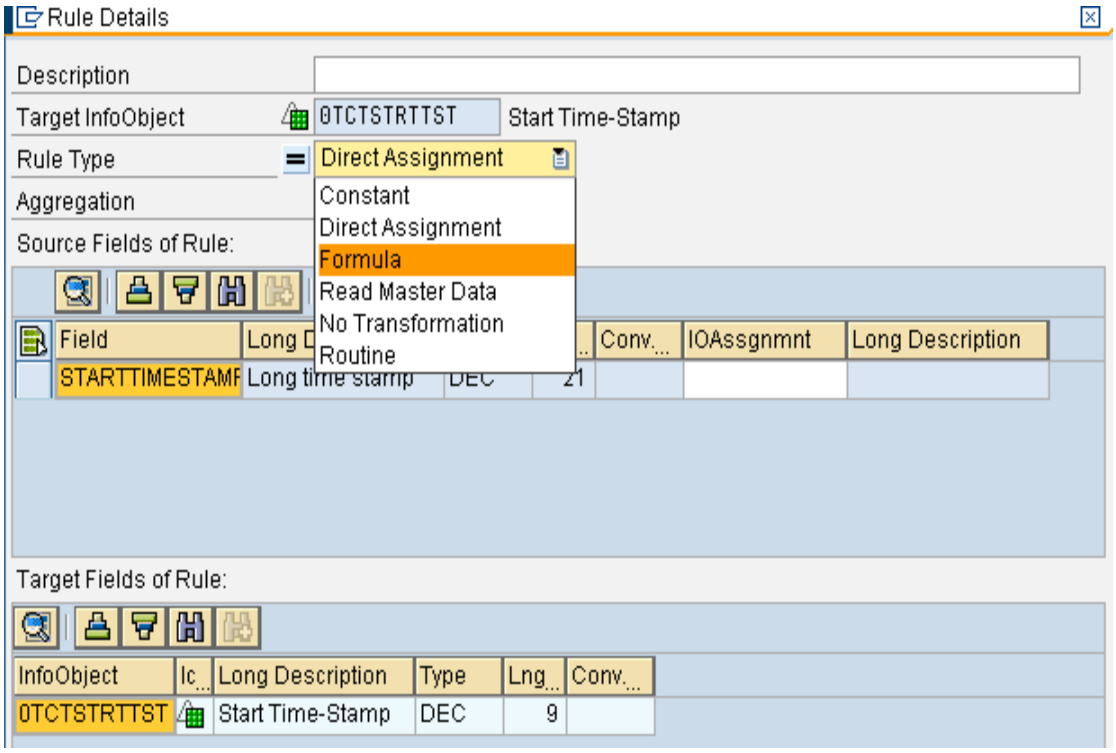
© 2011 SAP AG

Step 4 – Checking in Formula Builder

1. To access your custom formulas, enter the Formula Builder either from your transformation, update rules or transfer rules.
   In our example, we have taken a transformation (shown below)
   And we will assign formula to the field (marked in red below)

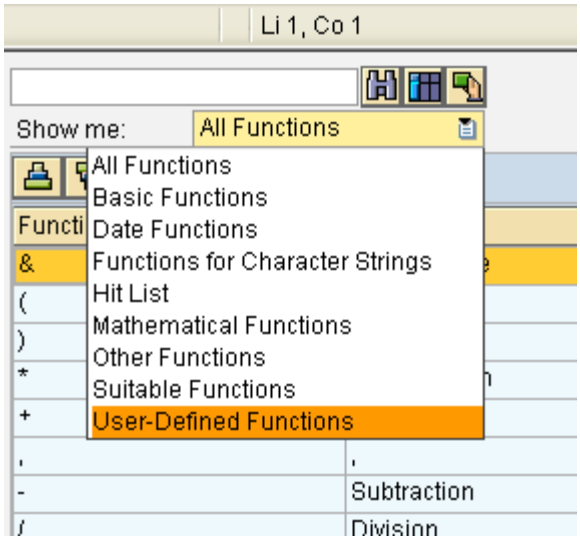


2. Click on the button before the field
3. New screen will appear like below, select Formula option

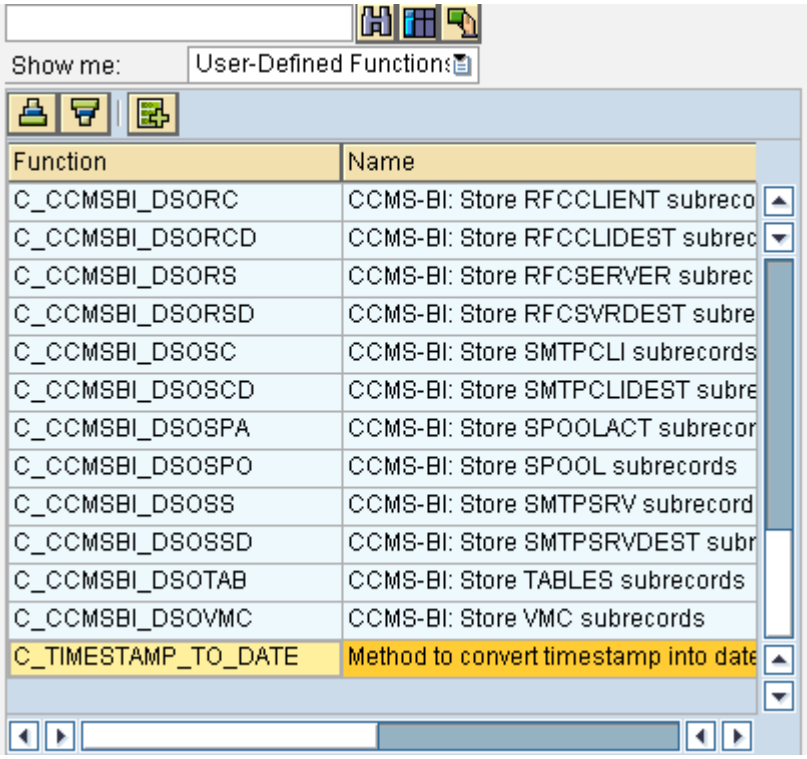


© 2011 SAP AG

4. Formula editor window will open with the list of formula categories
   Select User-Defined Functions



5. In the list of User defined formula , select our formula C_TIMESTAMP_TO_DATE



6. Implement this Function in our Transformation.

Conclusion

Thus, the Custom formulas are integrated into Formula builders.

© 2011 SAP AG

Related Content

BAdI: Customer-Defined Functions in the Formula Builder

The Transformation Library and Formula Builder

For more information, visit the EDW homepage

© 2011 SAP AG

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.

© 2011 SAP AG